Лабораторная работа №4.

Линейная алгебра.

Ишанова А.И. группа НФИ-02-19

Содержание

List of Figures

1 Цель работы

Изучение возможностей специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

2 Задание

1. Используя Jupyter Lab, повторите примеры из раздела 4.2.
2. Выполните задания для самостоятельной работы (раздел 4.4). [1]

3 Выполнение лабораторной работы

3.1 Повторение примеров

1. Повторяем примеры с поэлементными операциями над многомерными массивами. (fig. 3.1 - fig. 3.2)

```
# Массив 4x3 со случайными целыми числами (от 1 до 20):
  a = rand(1:20,(4,3))
```

```
4×3 Matrix{Int64}:
 8  18   6
 1  18  15
 9   1   8
 1   9  12
```

```
# Поэлементная сумма:
sum(a)
```

```
106
```

```
# Поэлементная сумма по столбцам:
sum(a,dims=1)
```

```
1×3 Matrix{Int64}:
 19  46  41
```

```
# Поэлементная сумма по строкам:
sum(a,dims=2)
```

```
4×1 Matrix{Int64}:
 32
 34
 18
 22
```

```
# Поэлементное произведение:
prod(a)
```

```
1813985280
```

```
# Поэлементное произведение по столбцам:
prod(a,dims=1)
```

```
1×3 Matrix{Int64}:
 72  2916  8640
```

```
# Поэлементное произведение по строкам:
prod(a,dims=2)
```

```
4×1 Matrix{Int64}:
 864
 270
  72
 108
```

Figure 3.1: Примеры с поэлементными операциями над многомерными массивами - 1

```
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics
```

```
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
```

```
# Вычисление среднего значения массива:
mean(a)
```

```
8.833333333333334
```

```
# Среднее по столбцам:
mean(a,dims=1)
```

```
1×3 Matrix{Float64}:
 4.75  11.5  10.25
```

```
# Среднее по строкам:
mean(a,dims=2)
```

```
4×1 Matrix{Float64}:
 10.666666666666666
 11.333333333333334
  6.0
  7.333333333333333
```

Figure 3.2: Примеры с поэлементными операциями над многомерными массивами - 2

2. Повторяем примеры с операциями над матрицами с пакетом LinearAlgebra (fig. 3.3 - fig. 3.4)

```
# Подключение пакета LinearAlgebra:
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
```

```
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
```

```
# Массив 4x4 со случайными целыми числами (от 1 до 20):
b = rand(1:20,(4,4))
```

```
4×4 Matrix{Int64}:
 15  19  17  13
 18  18   4   8
  2   9  10   5
 20  19   3  20
```

```
# Транспонирование:
transpose(b)
```

```
4×4 transpose(::Matrix{Int64}) with eltype Int64:
 15  18   2  20
 19  18   9  19
 17   4  10   3
 13   8   5  20
```

```
# След матрицы (сумма диагональных элементов):
tr(b)
```

```
63
```

```
# Извлечение диагональных элементов как массив:
diag(b)
```

```
4-element Vector{Int64}:
 15
 18
 10
 20
```

```
# Ранг матрицы:
rank(b)
```

```
4
```

```
# Инверсия матрицы (определение обратной матрицы):
inv(b)
```

```
4×4 Matrix{Float64}:
  0.138926      0.00403757  -0.227258   -0.0351022
 -0.166282      0.104895     0.238794    0.00642716
  0.121457     -0.0468853   -0.0753131  -0.0413645
  0.000823995  -0.0966546    0.0117007   0.0852011
```

Figure 3.3: Примеры с операциями над матрицами с пакетом LinearAlgebra - 1

```
# Определитель матрицы:
det(b)
```

```
12136.000000000002
```

```
# Псевдобратная функция для прямоугольных матриц:
pinv(a)
```

```
3×4 Matrix{Float64}:
  0.0481849  -0.0405044   0.0754184  -0.023741
  0.049925    0.0153562  -0.0444718  -0.0145098
 -0.0548958   0.0285036   0.0402613   0.0483108
```

Figure 3.4: Примеры с операциями над матрицами с пакетом LinearAlgebra - 2

3. Повторяем примеры из раздела “Вычисление нормы векторов и матриц, повороты, вращения”. (fig. 3.5 - fig. 3.7)

```
# Создание вектора X:
X = [2, 4, -5]

# Вычисление евклидовой нормы:
norm(X)
```

```
6.708203932499369
```

```
# Вычисление p-нормы:
p=1
norm(X,p)
```

```
11.0
```

```
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
norm(X-Y)
```

```
9.486832980505138
```

```
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))
```

```
9.486832980505138
```

```
# Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))
```

```
2.4404307889469252
```

Figure 3.5: Примеры из раздела “Вычисление нормы векторов и матриц, повороты, вращения” - 1

```
# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
# Вычисление Евклидовой нормы:
opnorm(d)
```

```
7.147682841795258
```

```
# Вычисление p-нормы:
p=1
opnorm(d,p)
```

```
8.0
```

```
# Поворот на 180 градусов:
rot180(d)
```

```
3×3 Matrix{Int64}:
 0   1  -2
 3   2  -1
 2  -4   5
```

```
# Переворачивание строк:
reverse(d,dims=1)
```

```
3×3 Matrix{Int64}:
 -2   1  0
 -1   2  3
  5  -4  2
```

```
# Переворачивание столбцов
reverse(d,dims=2)
```

```
3×3 Matrix{Int64}:
 2  -4   5
 3   2  -1
 0   1  -2
```

Figure 3.6: Примеры из раздела “Вычисление нормы векторов и матриц, повороты, вращения” - 2

4. Повторяем примеры из раздела “Матричное умножение, единичная матрица, скалярное произведение”. (fig. 3.7)

```
# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand(1:10,(2,3))
```

```
2×3 Matrix{Int64}:
 3  8  7
 3  2  9
```

```
# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand(1:10,(3,4))
```

```
3×4 Matrix{Int64}:
 7  4  6  2
 2  8  1  1
 4  8  2  7
```

```
# Произведение матриц A и B:
A*B
```

```
2×4 Matrix{Int64}:
 65  132  40  63
 61  100  38  71
```

```
# Единичная матрица 3x3:
Matrix{Int}(I, 3, 3)
```

```
3×3 Matrix{Int64}:
 1  0  0
 0  1  0
 0  0  1
```

```
# Скалярное произведение векторов X и Y:
X = [2, 4, -5]
Y = [1,-1,3]
dot(X,Y)
```

```
-17
```

```
# тоже скалярное произведение:
X'Y
```

```
-17
```

Figure 3.7: Примеры из раздела “Матричное умножение, единичная матрица, скалярное произведение”

5. Повторяем примеры из раздела “Факторизация. Специальные матричные структуры”. (fig. 3.8 - fig. 3.17)

```
# Задаём квадратную матрицу 3x3 со случайными значениями:
A = rand(3, 3)
```

```
3×3 Matrix{Float64}:
 0.484023  0.623866  0.271227
 0.434113  0.952469  0.125433
 0.345751  0.361731  0.28935
```

```
# Задаём единичный вектор:
x = fill(1.0, 3)
```

```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

```
# Задаём вектор b:
b = A*x
```

```
3-element Vector{Float64}:
 1.3791169552289078
 1.512013895917551
 0.9968322281932307
```

```
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x — единичный вектор):
A\b
```

```
3-element Vector{Float64}:
 1.0000000000000027
 0.999999999999999
 0.9999999999999982
```

Figure 3.8: Примеры из раздела "Факторизация. Специальные матричные структуры" - 1

```julia
# LU-факторизация:
Alu = lu(A)
```

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
 1.0        0.0       0.0
 0.896883   1.0       0.0
 0.714327  -0.213558  1.0
U factor:
3×3 Matrix{Float64}:
 0.484023  0.623866   0.271227
 0.0       0.392933  -0.117826
 0.0       0.0        0.0704426
```

```julia
# Матрица перестановок:
Alu.P
```

```
3×3 Matrix{Float64}:
 1.0  0.0  0.0
 0.0  1.0  0.0
 0.0  0.0  1.0
```

```julia
# Вектор перестановок:
Alu.p
```

```
3-element Vector{Int64}:
 1
 2
 3
```

```julia
# Матрица L:
Alu.L
```

```
3×3 Matrix{Float64}:
 1.0        0.0       0.0
 0.896883   1.0       0.0
 0.714327  -0.213558  1.0
```

```julia
# Матрица U:
Alu.U
```

```
3×3 Matrix{Float64}:
 0.484023  0.623866   0.271227
 0.0       0.392933  -0.117826
 0.0       0.0        0.0704426
```

Figure 3.9: Примеры из раздела “Факторизация. Специальные матричные структуры” - 2

```
# Решение СЛАУ через матрицу A:
A\b
```

```
3-element Vector{Float64}:
 1.0000000000000027
 0.999999999999999
 0.9999999999999982
```

```
# Решение СЛАУ через объект факторизации:
Alu\b
```

```
3-element Vector{Float64}:
 1.0000000000000027
 0.999999999999999
 0.9999999999999982
```

```
# Детерминант матрицы A:
det(A)
```

```
0.013397407753057599
```

```
# Детерминант матрицы A через объект факторизации:
det(Alu)
```

```
0.013397407753057599
```

Figure 3.10: Примеры из раздела "Факторизация. Специальные матричные структуры" - 3

```
# QR-факторизация:
Aqr = qr(A)
```

```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor:
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.657289   0.358133  -0.663108
 -0.589511  -0.792489   0.156328
 -0.469519   0.493662   0.732017
R factor:
3×3 Matrix{Float64}:
 -0.736394  -1.14139  -0.388074
  0.0       -0.35282   0.140573
  0.0        0.0       0.0515652
```

```
# Матрица Q:
Aqr.Q
```

```
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.657289   0.358133  -0.663108
 -0.589511  -0.792489   0.156328
 -0.469519   0.493662   0.732017
```

```
# Матрица R:
Aqr.R
```

```
3×3 Matrix{Float64}:
 -0.736394  -1.14139  -0.388074
  0.0       -0.35282   0.140573
  0.0        0.0       0.0515652
```

```
# Проверка, что матрица Q – ортогональная:
Aqr.Q'*Aqr.Q
```

```
3×3 Matrix{Float64}:
 1.0  -5.55112e-17  0.0
 0.0   1.0          1.66533e-16
 0.0   1.66533e-16  1.0
```

Figure 3.11: Примеры из раздела “Факторизация. Специальные матричные структуры” - 4

```
#  Симметризация матрицы A:
Asym = A + A'
```

```
3×3 Matrix{Float64}:
 0.968047  1.05798   0.616978
 1.05798   1.90494   0.487164
 0.616978  0.487164  0.578701
```

```
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
```

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 0.06486717396819534
 0.5453799725267454
 2.8414376012764384
vectors:
3×3 Matrix{Float64}:
 -0.72855    0.417193  -0.543291
  0.250053  -0.576421  -0.777954
  0.637721   0.70263   -0.315631
```

```
# Собственные значения:
AsymEig.values
```

```
3-element Vector{Float64}:
 0.06486717396819534
 0.5453799725267454
 2.8414376012764384
```

```
#Собственные векторы:
AsymEig.vectors
```

```
3×3 Matrix{Float64}:
 -0.72855    0.417193  -0.543291
  0.250053  -0.576421  -0.777954
  0.637721   0.70263   -0.315631
```

```
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym
```

```
3×3 Matrix{Float64}:
  1.0          -3.9968e-15   -1.33227e-15
 -6.66134e-16   1.0          -4.44089e-16
  8.88178e-16   2.22045e-15   1.0
```

Figure 3.12: Примеры из раздела “Факторизация. Специальные матричные структуры” - 5

```julia
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
```

```
1000×1000 Matrix{Float64}:
  1.09331     0.363573    0.496485   …  -1.76341    -0.29403    -1.50402
 -0.724628    0.791012    0.778186      -0.818687   0.738579   -1.50574
 -0.216552   -1.35119     0.736085       0.032596   3.08291     0.0295347
  0.163385   -0.297444    0.104863       1.54627    0.712209   -0.0278661
  0.994555    0.512395    1.12558       -0.403578  -0.866692    0.44531
  0.0811756   1.11385    -0.20963    …   1.57396   -0.677453    1.61112
 -0.773997   -0.172837   -1.05391       -0.0932662  1.94645    -1.01871
  2.52906    -0.901346   -0.843311       1.0383     0.623605    0.656359
  0.982829    0.265711    0.156347       0.352033  -0.284927   -0.382084
  1.11329    -1.40048     0.668007      -2.02452    0.50392    -0.961662
  0.383897    0.010126    0.0196763  …  -0.709709  -0.376469    0.815097
  0.0937731   1.4082     -2.4076         0.411469  -1.4196      1.60788
  0.843918    0.497192    1.56077       -0.0221146 -0.899836    0.47918
  ⋮                                  ⋱
 -0.0347134  -0.555417    2.42698       -0.461311   0.272944    0.972082
 -0.239573    0.0896085  -0.828352      -1.05095   -1.51708     1.20302
  0.415476   -0.415669   -0.221218   …  -0.121232  -0.436856    0.00618963
 -0.952634    1.23785     0.823623      -0.333718  -0.16586     0.46349
 -0.0803476   1.26081     0.772363       1.82774   -1.11934     0.0655876
  0.842054    0.568115    0.34688       -1.83662    0.925081   -0.00218425
  0.244448    0.337534    0.839703       1.60527   -1.308      -0.324341
 -0.353554    0.76102     0.511991   …   0.15865   -0.224898    1.67484
 -1.72469    -0.652394   -0.765574      -2.09769    0.0897537   0.198026
 -0.730816   -0.684269   -0.493971      -0.438051  -0.859559   -0.797776
  1.48212    -1.73081    -0.27274        1.15672    0.0654311   0.960161
 -0.530268   -0.500795   -0.496474      -0.0215955  0.261902   -1.95928
```

```julia
# Симметризация матрицы:
Asym = A + A'
```

```
1000×1000 Matrix{Float64}:
  2.18661   -0.361055    0.279933  …  -2.49423    1.18809    -2.03428
 -0.361055   1.58202    -0.573        -1.50296   -0.992234   -2.00654
  0.279933  -0.573       1.47217      -0.461375   2.81017    -0.466939
  0.668439  -0.402568   -0.48608       1.92255    0.643016   -0.887874
  1.71045   -0.390169    0.562203      0.104965   0.208191    1.59537
  0.885325   2.61484     0.451158  …   0.423864  -1.47572     2.71705
 -0.970016  -1.06746    -0.780918      0.213899   2.78338    -1.24366
  3.69698   -1.33448    -2.50497       1.23476   -1.14255     2.56779
 -0.210032  -0.998559    0.532328     -1.22678   -0.453834   -1.05629
  2.72342   -2.67369     3.0387       -0.537171   0.612701    0.672373
  0.226774  -0.199525   -0.535861  …  -2.98722   -1.32099    -0.0612656
 -0.833087   1.09618    -2.63641       1.80995   -2.41069     1.49708
  1.43679    0.112258    1.38431       0.264252  -0.810277   -0.813267
  ⋮                                ⋱
 -0.761554  -1.25494     0.4716       -1.33875   -0.538068    0.559707
 -0.529632  -0.800666   -1.72588      -0.633209  -1.31007     0.616118
  0.752629  -0.0617439  -0.922474  …  -0.854399   0.0675425   1.30087
 -2.25919    3.08421     1.3232        0.323439   0.511136   -1.20275
 -1.33657    1.28897     0.421992      0.107038  -0.238649   -0.802519
  1.60278    0.528414    1.13165      -2.32809    2.14216     0.988029
  0.487394   0.479831    0.165436      1.56637   -0.0624694  -0.727635
```

Figure 3.13: Примеры из раздела "Факторизация. Специальные матричные структуры" - 6

```
# Проверка, является ли матрица симметричной:
issymmetric(Asym)
```

```
true
```

```
# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
```

```
-0.36105516619829875
```

```
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)
```

```
false
```

```
# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
```

```
1000×1000 Symmetric{Float64, Matrix{Float64}}:
  2.18661   -0.361055    0.279933  …  -2.49423    1.18809    -2.03428
 -0.361055   1.58202    -0.573        -1.50296   -0.992234   -2.00654
  0.279933  -0.573       1.47217      -0.461375   2.81017    -0.466939
  0.668439  -0.402568   -0.48608       1.92255    0.643016   -0.887874
  1.71045   -0.390169    0.562203      0.104965   0.208191    1.59537
  0.885325   2.61484     0.451158  …   0.423864  -1.47572     2.71705
 -0.970016  -1.06746    -0.780918      0.213899   2.78338    -1.24366
  3.69698   -1.33448    -2.50497       1.23476   -1.14255     2.56779
 -0.210032  -0.998559    0.532328     -1.22678   -0.453834   -1.05629
  2.72342   -2.67369     3.0387       -0.537171   0.612701    0.672373
  0.226774  -0.199525   -0.535861  …  -2.98722   -1.32099    -0.0612656
 -0.833087   1.09618    -2.63641       1.80995   -2.41069     1.49708
  1.43679    0.112258    1.38431       0.264252  -0.810277   -0.813267
  ⋮                                ⋱
 -0.761554  -1.25494     0.4716       -1.33875   -0.538068    0.559707
 -0.529632  -0.800666   -1.72588      -0.633209  -1.31007     0.616118
  0.752629  -0.0617439  -0.922474  …  -0.854399   0.0675425   1.30087
 -2.25919    3.08421     1.3232        0.323439   0.511136   -1.20275
 -1.33657    1.28897     0.421992      0.107038  -0.238649   -0.802519
  1.60278    0.528414    1.13165      -2.32809    2.14216     0.988029
  0.487394   0.479831    0.165436      1.56637   -0.0624694  -0.727635
  2.46724    0.33127    -0.194843  …  -0.594897   1.3797      1.33205
 -1.37197   -1.24193    -2.33148      -1.70881    0.71292     0.404588
 -2.49423   -1.50296    -0.461375     -0.876102   0.297163   -0.819371
  1.18809   -0.992234    2.81017       0.297163   0.130862    1.22206
 -2.03428   -2.00654    -0.466939     -0.819371   1.22206    -3.91856
```

Figure 3.14: Примеры из раздела “Факторизация. Специальные матричные структуры” - 7

```julia
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
```

```
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
```

```julia
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);
```

```
  76.094 ms (11 allocations: 7.99 MiB)
```

```julia
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы:
@btime eigvals(Asym_noisy);
```

```
  618.973 ms (13 allocations: 7.92 MiB)
```

```julia
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы,
# для которой явно указано, что она симметричная:
@btime eigvals(Asym_explicit);
```

```
  74.211 ms (11 allocations: 7.99 MiB)
```

Figure 3.15: Примеры из раздела “Факторизация. Специальные матричные структуры” - 8

```julia
# Трёхдиагональная матрица 1000000 x 1000000:
n = 1000000;
A = SymTridiagonal(randn(n), randn(n-1))
```

```
1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
 -0.156001  -1.07412     ⋅        …   ⋅          ⋅          ⋅
 -1.07412   -0.740443   0.651895      ⋅          ⋅          ⋅
   ⋅         0.651895  -0.401327      ⋅          ⋅          ⋅
   ⋅          ⋅         0.30778       ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅        …   ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅        …   ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
  ⋮                               ⋱
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅        …   ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅            ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅        …   ⋅          ⋅          ⋅
   ⋅          ⋅          ⋅           1.23595     ⋅          ⋅
   ⋅          ⋅          ⋅           0.404675  -1.42635     ⋅
   ⋅          ⋅          ⋅          -1.42635    0.978342  -1.07449
   ⋅          ⋅          ⋅            ⋅        -1.07449   -0.0469415
```

```julia
# Оценка эффективности выполнения операции по нахождению
# собственных значений:
@btime eigmax(A)
```

```
  547.812 ms (17 allocations: 183.11 MiB)
6.514227997785152
```

Figure 3.16: Примеры из раздела “Факторизация. Специальные матричные структуры” - 9

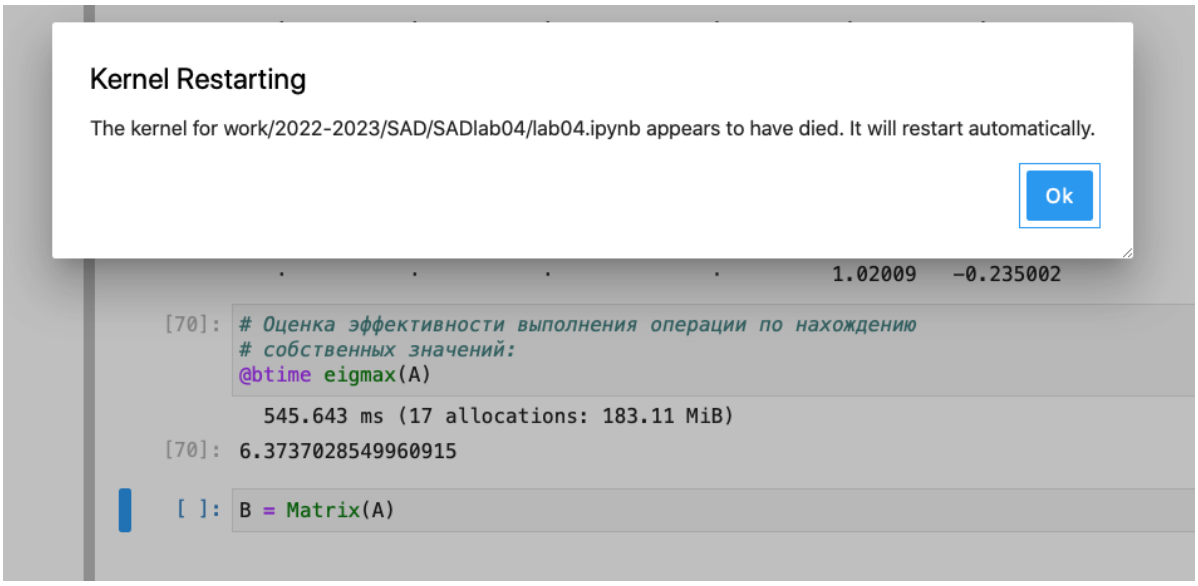

Figure 3.17: Ошибка при выполнении команды B = Matrix(A)

6. Повторяем примеры из раздела “Общая линейная алгебра”. (fig. 3.18)

```
# Матрица с рациональными элементами:
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
```

```
3×3 Matrix{Rational{BigInt}}:
 3//5  3//10  3//5
 4//5  1//5   7//10
 4//5  9//10  4//5
```

```
# Единичный вектор:
x = fill(1, 3)
```

```
3-element Vector{Int64}:
 1
 1
 1
```

```
# Задаём вектор b:
b = Arational*x
```

```
3-element Vector{Rational{BigInt}}:
  3//2
 17//10
  5//2
```

```
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x – единичный вектор):
Arational\b
```

```
3-element Vector{Rational{BigInt}}:
 1//1
 1//1
 1//1
```

```
# LU-разложение:
lu(Arational)
```

```
LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3×3 Matrix{Rational{BigInt}}:
 1//1  0//1   0//1
 1//1  1//1   0//1
 3//4  3//14  1//1
U factor:
3×3 Matrix{Rational{BigInt}}:
 4//5  1//5   7//10
 0//1  7//10  1//10
 0//1  0//1   3//56
```

Figure 3.18: Примеры из раздела “Общая линейная алгебра”

3.2 Задания для самостоятельной работы

1. Призведение векторов.

1.1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результат в dot_v. (fig. 3.19)

1.2. Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной outer_v. (fig. 3.19)

```
# ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ
```

```
# 1.1

v = [1, 2, 3, 4, 5]
dot_v = v'v
```

```
55
```

```
# 1.2
outer_v = v * v'
```

```
5×5 Matrix{Int64}:
 1   2   3   4   5
 2   4   6   8  10
 3   6   9  12  15
 4   8  12  16  20
 5  10  15  20  25
```

Figure 3.19: Скалярное и матричное умножение вектора на себя

2. Системы линейных уравнений.

2.1. Решить СЛАУ с двумя неизвестными.

a) (fig. 3.20)

$$\begin{cases} x + y = 2, \\ x - y = 3. \end{cases}$$

b) Система имеет бесконечно много решений, поэтому программа выдает ошибку. (fig. 3.20)

$$\begin{cases} x + y = 2, \\ 2x + 2y = 4. \end{cases}$$

c) Система не имеет решений, поэтому программа выдает ошибку. (fig. 3.20)

$$\begin{cases} x + y = 2, \\ 2x + 2y = 5. \end{cases}$$

d) (fig. 3.21)

$$\begin{cases} x + y = 1, \\ 2x + 2y = 2, \\ 3x + 3y = 3. \end{cases}$$

e) (fig. 3.21)

$$\begin{cases} x + y = 2, \\ 2x + y = 1, \\ x - y = 3. \end{cases}$$

f) (fig. 3.21)

$$\begin{cases} x + y = 2, \\ 2x + y = 1, \\ 3x + 2y = 3. \end{cases}$$

```
#2.1

A = [ 1 1; 1 -1]
b = [2;3]
A\b
```

```
2-element Vector{Float64}:
  2.5
 -0.5
```

```
A = [1 1; 2 2]
b = [4;2]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:19 [inlined]
 [2] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:21 [inlined]
 [3] #lu!#170
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:82 [inlined]
 [4] #lu#177
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:279 [inlined]
 [5] lu (repeats 2 times)
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:278 [inlined]
 [6] \(A::Matrix{Int64}, B::Vector{Int64})
   @ LinearAlgebra /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/Lin
earAlgebra/src/generic.jl:1110
 [7] top-level scope
   @ In[86]:3
 [8] eval
   @ ./boot.jl:368 [inlined]
 [9] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1428
```

```
A = [1 1; 2 2]
b = [2;5]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:19 [inlined]
 [2] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:21 [inlined]
 [3] #lu!#170
```

Figure 3.20: Решение СЛАУ с двумя неизвестными - 1

```
A = [1 1; 2 2; 3 3]
b = [1; 2; 3]
A\b
```

```
2-element Vector{Float64}:
 0.4999999999999999
 0.5
```

```
A = [1 1; 2 1; 1 -1]
b = [2; 1; 3]
A\b
```

```
2-element Vector{Float64}:
  1.5000000000000004
 -0.9999999999999997
```

```
A = [1 1; 2 1; 3 2]
b = [2; 1; 3]
A\b
```

```
2-element Vector{Float64}:
 -0.9999999999999989
  2.9999999999999982
```

Figure 3.21: Решение СЛАУ с двумя неизвестными - 2

2.2. Решить СЛАУ с тремя неизвестными.

a) (fig. 3.22)

$$\begin{cases} x + y + z = 2, \\ x - y - 2z = 3. \end{cases}$$

b) (fig. 3.22)

$$\begin{cases} x + y + z = 2, \\ 2x + 2y - 3z = 4, \\ 3x + y + z = 1. \end{cases}$$

c) Система имеет бесконечно много решений, поэтому программа выдает ошибку.(fig. 3.22)

$$\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 1. \end{cases}$$

d) Система не имеет решений, поэтому программа выдает ошибку. (fig. 3.23)

$$\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 0. \end{cases}$$

```
# 2.2

A = [1 1 1; 1 -1 -2]
b = [2; 3]
A\b
```

```
3-element Vector{Float64}:
  2.2142857142857144
  0.35714285714285704
 -0.5714285714285712
```

```
A = [1 1 1; 2 2 -3; 3 1 1]
b = [2; 4; 1]
A\b
```

```
3-element Vector{Float64}:
 -0.5
  2.5
  0.0
```

```
A = [1 1 1; 1 1 2; 2 2 3]
b = [1; 0; 1]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:19 [inlined]
 [2] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:21 [inlined]
 [3] #lu!#170
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:82 [inlined]
 [4] #lu#177
```

Figure 3.22: Решение СЛАУ с тремя неизвестными - 1

```
A = [1 1 1; 1 1 2; 2 2 3]
b = [1; 0; 0]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:19 [inlined]
 [2] checknonsingular
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/factorization.jl:21 [inlined]
 [3] #lu!#170
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:82 [inlined]
 [4] #lu#177
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:279 [inlined]
 [5] lu (repeats 2 times)
   @ /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/LinearAlgebra/src
/lu.jl:278 [inlined]
 [6] \(A::Matrix{Int64}, B::Vector{Int64})
   @ LinearAlgebra /Applications/Julia-1.8.app/Contents/Resources/julia/share/julia/stdlib/v1.8/Lin
earAlgebra/src/generic.jl:1110
 [7] top-level scope
   @ In[98]:3
 [8] eval
   @ ./boot.jl:368 [inlined]
 [9] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1428
```

Figure 3.23: Решение СЛАУ с тремя неизвестными - 2

3. Операции с матрицами.

3.1. Приведите матрицы к диагональному виду.

a) (fig. 3.24)

$$\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$$

b) (fig. 3.24)

$$\begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}$$

c) (fig. 3.24)

$$\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix}$$

```
# 3.1
M = [1 -2; -2 1]
Diagonal(eigen(M).values)
```

```
2×2 Diagonal{Float64, Vector{Float64}}:
 -1.0   ⋅
   ⋅   3.0
```

```
Matrix(Diagonal(eigen(M).values))
```

```
2×2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
```

```
M = [1 -2; -2 3]
Diagonal(eigen(M).values)
```

```
2×2 Diagonal{Float64, Vector{Float64}}:
 -0.236068   ⋅
   ⋅        4.23607
```

```
M = [1 -2 0; -2 1 2; 0 2 0]
Diagonal(eigen(M).values)
```

```
3×3 Diagonal{Float64, Vector{Float64}}:
 -2.14134   ⋅         ⋅
   ⋅       0.515138   ⋅
   ⋅        ⋅        3.6262
```

Figure 3.24: Приведение матриц к диагональному виду

3.2. Вычислить:

a) (fig. 3.25)

$$\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}^{10}$$

b) (fig. 3.25)

$$\sqrt{\begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}}$$

c) (fig. 3.25)

$$\sqrt[3]{\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}}$$

d) (fig. 3.25)

$$\sqrt{\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}}$$

```
# 3.2

C = [1 -2; -2 1]
C^10
```

```
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
```

```
C = [5 -2; -2 5]
sqrt(C)
```

```
2×2 Matrix{Float64}:
  2.1889   -0.45685
 -0.45685   2.1889
```

```
C = [1 -2; -2 1]
C^(1//3)
```

```
2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
  0.971125+0.433013im  -0.471125+0.433013im
 -0.471125+0.433013im   0.971125+0.433013im
```

```
C = [1 2; 2 3]
sqrt(C)
```

```
2×2 Matrix{ComplexF64}:
 0.568864+0.351578im  0.920442-0.217287im
 0.920442-0.217287im   1.48931+0.134291im
```

Figure 3.25: Вычисление операций над матрицами

3.3. Найти собственные значения, создать диагональную матрицу из собственных значений и нижнедиагональную матрицу из исходной, оценить эффективность выполнения операций. (fig. 3.26)

$$A = \begin{pmatrix} 140 & 97 & 74 & 168 & 131 \\ 97 & 106 & 89 & 131 & 36 \\ 74 & 89 & 152 & 144 & 71 \\ 168 & 131 & 144 & 54 & 142 \\ 131 & 36 & 71 & 142 & 36 \end{pmatrix}$$

```
#3.3

A = [140 97 74 168 131;
    97 106 89 131 36;
    74 89 152 144 71;
    168 131 144 54 142;
    131 36 71 142 36]
```

```
5×5 Matrix{Int64}:
 140   97   74  168  131
  97  106   89  131   36
  74   89  152  144   71
 168  131  144   54  142
 131   36   71  142   36
```

```
eigvals(A)
```

```
3-element Vector{Float64}:
 0.02679491924311228
 0.1
 0.37320508075688774
```

```
Diagonal(eigen(A).values)
```

```
3×3 Diagonal{Float64, Vector{Float64}}:
 0.0267949   ⋅    ⋅
  ⋅         0.1   ⋅
  ⋅          ⋅   0.373205
```

```
lu(A).L
```

```
5×5 Matrix{Float64}:
 1.0        0.0        0.0        0.0       0.0
 0.779762   1.0        0.0        0.0       0.0
 0.440476  -0.47314    1.0        0.0       0.0
 0.833333   0.183929  -0.556312   1.0       0.0
 0.577381  -0.459012  -0.189658   0.897068  1.0
```

```
@btime eigvals(A);
```

```
  1.458 μs (12 allocations: 1.64 KiB)
```

```
@btime Diagonal(eigen(A).values);
```

```
  4.289 μs (18 allocations: 4.28 KiB)
```

```
@btime lu(A).L;
```

```
  601.695 ns (4 allocations: 640 bytes)
```

Figure 3.26: Нахождение собственных значений, составление диагональной матрицы из собственных значений, составление нижнедтиагонально матрицы и оценивание эффективности этих операций.

4. Линейные модели экономики.

Линейная модель:

$$x - Ax = y$$

Проверить являются ли матрицы A продуктивными по критерию:

4.1. Критерий: Решение системы x при любом неотрицательном y имеет только неотрицательные элементы.

a) (fig. 3.27)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

b)(fig. 3.27)

$$\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

c)(fig. 3.27)

$$\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
# 4.1

A = [1 2; 3 4]
E = Matrix{Int}(I, 2, 2)
y = rand(0:10000, 2)
all(>=(0), y\(E-A))
```

```
false
```

```
A = Matrix{Rational{Int}}([1 2; 3 4])/2
y = rand(0:10000, 2)
all(>=(0), y\(E-A))
```

```
false
```

```
A = Matrix{Rational{Int}}([1 2; 3 4])/10
y = rand(0:10000, 2)
all(>=(0), y\(E-A))
```

```
true
```

Figure 3.27: Проверка продуктивности матрицы A по критерию 1

4.2. Все элементы матрицы $(E - A)^{-1}$ - неотрицательные числа.

a) (fig. 3.28)

$$\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

b)(fig. 3.28)

$$\frac{1}{2}\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}$$

c)(fig. 3.28)

$$\frac{1}{10}\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}$$

```
# 4.2

A = [1 2; 3 1]
all(>=(0), inv(E-A))
```

```
false
```

```
A = Matrix{Rational{Int}}([1 2; 3 1])/2
all(>=(0), inv(E-A))
```

```
false
```

```
A = Matrix{Rational{Int}}([1 2; 3 1])/10
all(>=(0), inv(E-A))
```

```
true
```

Figure 3.28: Проверка продуктивности матрицы A по критерию 2

4.3. Все собственные значения матрицы A по модулю меньше 1.

a) (fig. 3.29)

$$\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}$$

b)(fig. 3.29)

$$\frac{1}{2}\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}$$

c)(fig. 3.29)

$$\frac{1}{10}\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}$$

d)(fig. 3.29)

$$\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0 & 0.1 & 0.2 \\ 0 & 0.1 & 0.3 \end{pmatrix}$$

```julia
# 4.3
A = [1 2; 3 1]
all(<(1), broadcast(abs,eigvals(A)))
```

```
false
```

```julia
A = Matrix{Rational{Int}}([1 2; 3 1])/2
all(<(1), broadcast(abs,eigvals(A)))
```

```
false
```

```julia
A = Matrix{Rational{Int}}([1 2; 3 1])/10
all(<(1), broadcast(abs,eigvals(A)))
```

```
true
```

```julia
A = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
all(<(1), broadcast(abs,eigvals(A)))
```

```
true
```

Figure 3.29: Проверка продуктивности матрицы A по критерию 3

4 Листинг

```
# -*- coding: utf-8 -*-
# ---
# jupyter:
#   jupytext:
#     text_representation:
#       extension: .jl
#       format_name: light
#       format_version: '1.5'
#       jupytext_version: 1.14.1
#   kernelspec:
#     display_name: Julia 1.8.2
#     language: julia
#     name: julia-1.8
# ---

# Массив 4x3 со случайными целыми числами (от 1 до 20):
  a = rand(1:20,(4,3))

# Поэлементная сумма:
sum(a)

# Поэлементная сумма по столбцам:
```

```
sum(a,dims=1)

# Поэлементная сумма по строкам:
sum(a,dims=2)

# Поэлементное произведение:
prod(a)

# Поэлементное произведение по столбцам:
prod(a,dims=1)

# Поэлементное произведение по строкам:
prod(a,dims=2)

 # Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics

# Вычисление среднего значения массива:
mean(a)

# Среднее по столбцам:
mean(a,dims=1)

# Среднее по строкам:
mean(a,dims=2)

# Подключение пакета LinearAlgebra:
```

```
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra

# Массив 4x4 со случайными целыми числами (от 1 до 20):
b = rand(1:20,(4,4))

# Транспонирование:
transpose(b)

# След матрицы (сумма диагональных элементов):
tr(b)

# Извлечение диагональных элементов как массив:
diag(b)

# Ранг матрицы:
rank(b)

# Инверсия матрицы (определение обратной матрицы):
inv(b)

# Определитель матрицы:
det(b)

# Псевдобратная функция для прямоугольных матриц:
pinv(a)

# +
```

```
# Создание вектора X:
X = [2, 4, -5]

# Вычисление евклидовой нормы:
norm(X)
# -

# Вычисление p-нормы:
p=1
norm(X,p)

# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
norm(X-Y)

# Проверка по базовому определению:
sqrt(sum((X-Y).^2))

# Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))

# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
# Вычисление Евклидовой нормы:
opnorm(d)

# Вычисление p-нормы:
p=1
```

```
opnorm(d,p)

# Поворот на 180 градусов:
rot180(d)

# Переворачивание строк:
reverse(d,dims=1)

# Переворачивание столбцов
reverse(d,dims=2)

# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand(1:10,(2,3))

# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand(1:10,(3,4))

# Произведение матриц A и B:
A*B

# Единичная матрица 3x3:
Matrix{Int}(I, 3, 3)

# Скалярное произведение векторов X и Y:
X = [2, 4, -5]
Y = [1,-1,3]
dot(X,Y)

# тоже скалярное произведение:
```

```
X'Y

# Задаём квадратную матрицу 3x3 со случайными значениями:
A = rand(3, 3)

# Задаём единичный вектор:
x = fill(1.0, 3)

# Задаём вектор b:
b = A*x

# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
A\b

# LU-факторизация:
Alu = lu(A)

# Матрица перестановок:
Alu.P

# Вектор перестановок:
Alu.p

# Матрица L:
Alu.L

# Матрица U:
Alu.U
```

```
# Решение СЛАУ через матрицу A:
A\b

# Решение СЛАУ через объект факторизации:
Alu\b

# Детерминант матрицы A:
det(A)

# Детерминант матрицы A через объект факторизации:
det(Alu)

# QR-факторизация:
Aqr = qr(A)

# Матрица Q:
Aqr.Q

# Матрица R:
Aqr.R

# Проверка, что матрица Q - ортогональная:
Aqr.Q'*Aqr.Q

#  Симметризация матрицы A:
Asym = A + A'

# Спектральное разложение симметризованной матрицы:
```

```
AsymEig = eigen(Asym)

# Собственные значения:
AsymEig.values

#Собственные векторы:
AsymEig.vectors

# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym

# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)

# Симметризация матрицы:
Asym = A + A'

# Проверка, является ли матрица симметричной:
issymmetric(Asym)

# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()

# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)

# Явно указываем, что матрица является симметричной:
```

```
Asym_explicit = Symmetric(Asym_noisy)

import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools

# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);

# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы:
@btime eigvals(Asym_noisy);

# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы,
# для которой явно указано, что она симметричная:
@btime eigvals(Asym_explicit);

# Трёхдиагональная матрица 1000000 x 1000000:
n = 1000000;
A = SymTridiagonal(randn(n), randn(n-1))

# + tags=[]
# Оценка эффективности выполнения операции по нахождению
# собственных значений:
@btime eigmax(A)
# -
```

```
B = Matrix(A)

# Матрица с рациональными элементами:
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10

# Единичный вектор:
x = fill(1, 3)

# Задаём вектор b:
b = Arational*x

# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
Arational\b

# LU-разложение:
lu(Arational)

# +
# ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

# +
# 1.1

v = [1, 2, 3, 4, 5]
dot_v = v'v
# -

# 1.2
```

```
outer_v = v * v'


# +
#2.1

A = [ 1 1; 1 -1]
b = [2;3]
A\b
# -

A = [1 1; 2 2]
b = [4;2]
A\b

A = [1 1; 2 2]
b = [2;5]
A\b

A = [1 1; 2 2; 3 3]
b = [1; 2; 3]
A\b

A = [1 1; 2 1; 1 -1]
b = [2; 1; 3]
A\b

A = [1 1; 2 1; 3 2]
b = [2; 1; 3]
A\b
```

```
# +
# 2.2

A = [1 1 1; 1 -1 -2]
b = [2; 3]
A\b
# -

A = [1 1 1; 2 2 -3; 3 1 1]
b = [2; 4; 1]
A\b

A = [1 1 1; 1 1 2; 2 2 3]
b = [1; 0; 1]
A\b

A = [1 1 1; 1 1 2; 2 2 3]
b = [1; 0; 0]
A\b

# 3.1
M = [1 -2; -2 1]
Diagonal(eigen(M).values)

Matrix(Diagonal(eigen(M).values))

M = [1 -2; -2 3]
Diagonal(eigen(M).values)
```

```
M = [1 -2 0; -2 1 2; 0 2 0]
Diagonal(eigen(M).values)

# +
# 3.2

C = [1 -2; -2 1]
C^10
# -

C = [5 -2; -2 5]
sqrt(C)

C = [1 -2; -2 1]
C^(1//3)

C = [1 2; 2 3]
sqrt(C)

# +
#3.3

A = [140 97 74 168 131;
    97 106 89 131 36;
    74 89 152 144 71;
    168 131 144 54 142;
    131 36 71 142 36]
# -
```

```julia
eigvals(A)

Diagonal(eigen(A).values)

lu(A).L

@btime eigvals(A);

@btime Diagonal(eigen(A).values);

@btime lu(A).L;

# +
# 4.1

A = [1 2; 3 4]
E = Matrix{Int}(I, 2, 2)
y = rand(0:10000, 2)
all(>=(0), y\(E-A))
# -

A = Matrix{Rational{Int}}([1 2; 3 4])/2
y = rand(0:10000, 2)
all(>=(0), y\(E-A))

A = Matrix{Rational{Int}}([1 2; 3 4])/10
y = rand(0:10000, 2)
all(>=(0), y\(E-A))
```

```julia
# +
# 4.2

A = [1 2; 3 1]
all(>=(0), inv(E-A))
# -

A = Matrix{Rational{Int}}([1 2; 3 1])/2
all(>=(0), inv(E-A))

A = Matrix{Rational{Int}}([1 2; 3 1])/10
all(>=(0), inv(E-A))

# 4.3
A = [1 2; 3 1]
all(<(1), broadcast(abs,eigvals(A)))

A = Matrix{Rational{Int}}([1 2; 3 1])/2
all(<(1), broadcast(abs,eigvals(A)))

A = Matrix{Rational{Int}}([1 2; 3 1])/10
all(<(1), broadcast(abs,eigvals(A)))

A = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
all(<(1), broadcast(abs,eigvals(A)))
```

5 Вывод

В ходе выполнения лабораторной работы операции применимые в рамках задач линейной алгебры. Были изучены возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры. Также были поставлены пакеты LinearAlgebra, Statistics и BenchmarkTools.

6 Библиография

1. Методические материалы курса.